BGPStream: A Software Framework for Live and Historical BGP Data Analysis

BGPStream: A Software Framework for Live and Historical BGP Data Analysis

Chiara Orsini [1], Alistair King[1], Danilo Giordano[2], Vasileios Giotsas[1], Alberto Dainotti[1]
[1]CAIDA, UC San Diego
[2]Politecnico di Torino

ABSTRACT

We present BGPStream, an open-source software framework for the analysis of both historical and real-time Border Gateway Protocol (BGP) measurement data. Although BGP is a crucial operational component of the Internet infrastructure, and is the subject of research in the areas of Internet performance, security, topology, protocols, economics, etc., there is no efficient way of processing large amounts of distributed and/or live BGP measurement data. BGPStream fills this gap, enabling efficient investigation of events, rapid prototyping, and building complex tools and large-scale monitoring applications (e.g., detection of connectivity disruptions or BGP hijacking attacks). We discuss the goals and architecture of BGPStream. We apply the components of the framework to different scenarios, and we describe the development and deployment of complex services for global Internet monitoring that we built on top of it.

1. INTRODUCTION

We present BGPStream, an open-source software framework[1] for the analysis of historical and live Border Gateway Protocol (BGP) measurement data. Although BGP is a crucial operational component of the Internet infrastructure, and is the subject of fundamental research (in the areas of performance, security, topology, protocols, economy, etc.), there is no efficient and easy way of processing large amounts of BGP measurement data. BGPStream fills this gap by making available a set of APIs and tools for processing large amounts of live and historical data, thus supporting investigation of specific events, rapid prototyping, and building complex tools and efficient large-scale monitoring applications (e.g., detection of connectivity disruptions or BGP hijacking attacks). We discuss the goals and architecture of BGPStream and we show how the components of the framework can be used in different applicative scenarios.

[1]BGPStream is distributed with the GPL v2 license and is available at bgpstream.caida.org.

2. BACKGROUND

BGP Data at Router Level

The Border Gateway Protocol (BGP) is the de-facto standard inter-domain routing protocol for the Internet: its primary function is to exchange reachability information among Autonomous Systems (ASes) [52]. Each AS announces to the others, by means of BGP update messages, the routes to its local prefixes and the preferred routes learned from its neighbors. Such messages provide information about how a destination can be reached through an ordered list of AS hops, called an *AS path*.

A BGP router maintains this reachability information in the *Routing Information Base* (RIB) [52], which is structured in three sets:

- *Adj-RIBs-In*: routes learned from inbound update messages from its neighbors.
- *Loc-RIB*: routes selected from Adj-RIBs-In by applying local policies (e.g., shortest path, peering relationships with neighbors); the router will install these routes in its routing table to establish where to forward packets.
- *Adj-RIBs-Out*: routes selected from Loc-RIB, which the router will announce to its neighbors; for each neighbor the router creates a specific *Adj-RIB-Out* based on local policies (e.g., peering relationship).

BGP Data Collection

Some operators make BGP routing information from their routers available for monitoring, troubleshooting and research purposes. BGP *looking glasses* give users limited (e.g., read-only) access to a command line interface of a router, or allow them to download the ASCII

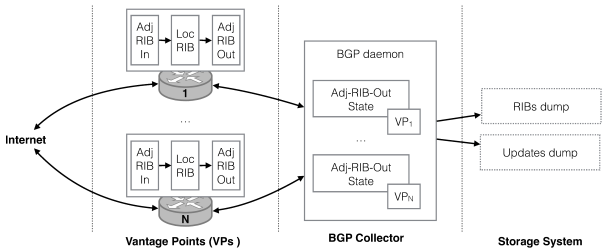


Figure 1: **BGP collection process.** Once a BGP collector establishes a BGP session with a VP, it maintains a state and an image of the VP's Adj-RIB-out table derived from the updates received through the session. With different periodicity, it dumps (i) a snapshot of all the VP Adj-RIB-out tables (RIB dump) and (ii) the update messages received within that period from all the VPs (Updates dump).

output of the current state of the router RIB. Looking glasses are more useful for interactive exploration rather than systematic and continuous data acquisition. The latter can instead be implemented either *(i)* by establishing a BGP peering session with the monitored router from a dedicated system (a *route collector*), or *(ii)* through a protocol specifically designed for monitoring purposes, such as OpenBMP [30,58]. OpenBMP is an open-source implementation of the BGP Monitoring Protocol defined in an IETF draft [58] and supported by latest versions of JunOS and Cisco IOS. The protocol allows a user to periodically access the Adj-RIBs-In of a router or to monitor its BGP peering sessions. While OpenBMP can be easily deployed within an AS to monitor its BGP routers, there are currently no projects which make such data publicly available. Route collectors instead, are often used for this purpose [48, 49, 56]. A route collector is a host running a collector process (e.g., Quagga [50]), which emulates a router and establishes BGP peering sessions with one or more real routers (*vantage points, VPs,* in the following). Each VP sends to the collector update messages (*updates*) each time the Adj-RIB-out changes, reflecting changes to its Loc-RIB (Figure 1).

Normally, a BGP session with a collector is configured as a *customer-provider* relationship, i.e., as if the VP was offering transit service to the collector. In this case, the VP is called *full-feed*, since it will advertise to the collector an Adj-RIB-Out which contains the entire set of routes in its Loc-RIB. This way, the collector potentially knows, at each instant, all the preferred-routes that the VP will use to reach the rest of the Internet – a partial view of the Internet topology graph visible to that router. A *partial-feed* VP instead, will provide through its Adj-RIB-Out only a subset of the routes in its Loc-RIB, e.g., routes to its own networks, or learned through its customers. Unfortunately, projects publicly providing information acquired by their collectors do not label VPs as full- or partial-feed, since peering with a collector is usually established on a voluntary basis and VP behavior can be subject to change without notice. Therefore, the policy that determines the Adj-RIB-Out to be shared with the collector must be dynamically inferred from the data (e.g., size of the Adj-RIB-Out).

For each VP, the collector maintains a session state and an image of the Adj-RIB-out table derived from updates. The collector periodically dumps, with a frequency of respectively few hours and few minutes, (i) a snapshot of the union of the maintained Adj-RIB-out tables (*RIB dump*) and (ii) the update messages received from all its VPs since the last dump, along with state changes (*Updates dump*). RIB dumps provide an efficient summary of changes to BGP routing tables with a coarse time granularity that is sufficient for several classes of studies [34, 42–44]. In contrast, Updates dumps carry a lot of information to be processed, but offer a complete view of the observable routing dynamics, enabling other types of analysis and near-realtime monitoring applications [35, 36, 46, 62].

Popular Data Sources

The most popular projects operating route collectors and making their dumps available in public archives are RouteViews [48] and RIPE RIS [56]. They currently operate 18 and 13 collectors respectively, which in total peer with approximately 380 and 600 VPs distributed worldwide (this number increases every year). Analyzing data from multiple VPs is of fundamental importance for most Internet studies, since each router has a limited view of the Internet topology and, even when full-feed, a VP shares only part of this information (the preferred routes). Moreover, macroscopic Internet phenomena visible through the routing infrastructure (e.g., outages, cyber attacks, peering relationships, performance issues, route leaks, router bugs) affect Internet routers differently, as a function of geography, topology, router operating system and hardware characteristics, operator, etc..

Such a distributed and detailed – even if partial – view of the inter-domain routing plane, generates large amounts of data (>2TB of compressed data collected in 2015 alone). RouteViews and RIPE RIS collectors save a RIB dump every 2 and 8 hours and an Updates dump every 15 and 5 minutes, respectively. Both projects save RIB and Updates dumps in a binary format, standardized by the IETF, called the Multi-Threaded Routing Toolkit (MRT) routing information export format [9]. RouteViews and RIPE RIS archives date back to 2001 and 1999 respectively, enabling longitudinal studies relevant to understand the evolution of the Internet infrastructure and its impact in other fields.

Software Frameworks and APIs

The most widely adopted software for BGP data analysis in the research community [5, 7, 18, 37, 53, 57, 60] is *libBGPdump* [54], an open source C library that provides a simple API to parse BGP dumps in MRT format and deserializes MRT records into custom data struc-

tures. It is distributed along with a command-line tool, *bgpdump*, that outputs MRT information read from a file in an ASCII format. Often researchers directly use the command-line tool to translate entire BGP dumps into text, and then parse the ASCII output to further process or archive the data. Although bgpdump has been an invaluable tool to support the analysis of BGP data over the last decade, it lacks the advanced features that we discuss in the next section (e.g., merging and sorting data from multiple files and data sources, supporting live processing, scalability, etc.).

There have been several projects that process BGP measurement data in real-time, developed both by industry (e.g., Dyn Research [28]) and academia (e.g., PHAS [41]), however their approaches are either undisclosed, or are specific to a certain application (i.e. they are not generalized frameworks). An exception is *BGPmon* [3, 63], a distributed monitoring system that retrieves BGP information by establishing BGP sessions with multiple ASes and that offers a live BGP data stream in the XML format (which also encapsulates the raw MRT data). Despite the fact that BGPmon enables rapid prototyping of live monitoring tools, it currently provides access to a limited number of VPs (compared to the vast number of VPs connected to RIS and RouteViews infrastructures), and it cannot be used for historical processing.

Towards Realtime Streaming of BGP Data

On the other hand, in the context of live monitoring, the major issue with popular public data sources such as RouteViews and RIPE RIS, is their file-based distribution system and thus the latency with which collected data is made available. Our measurements [24] show that, in addition to the 5 and 15 minutes delay due to file rotation duration, there is a small amount of variable delay due to publication infrastructure. However, 99% of Updates dumps in the last year were available in less than 20 minutes after the dump was begun. Since these latency values are low enough to enable several near-realtime monitoring applications, we began developing BGPStream with support for these data sources.

The research community recognizes the need for better support of live BGP measurement data collection and analysis. Since early 2015, we have been cooperating with other research groups and institutions (e.g., RouteViews, BGPMon, RIPE RIS) to coordinate efforts in this space [17]. Both RIPE RIS and BGPMon are developing a new BGP data streaming service (including investigating support for streamed MRT records), and BGPMon partners with RouteViews to include in the forthcoming next-generation BGPMon service all of their collectors. Experience with the development of BGPStream informed development efforts of the other research teams and vice-versa. While BGPStream is fully usable today, we envision that the forthcoming developments of these projects, likely deployed in 2016, will enhance BGPStream capabilities.

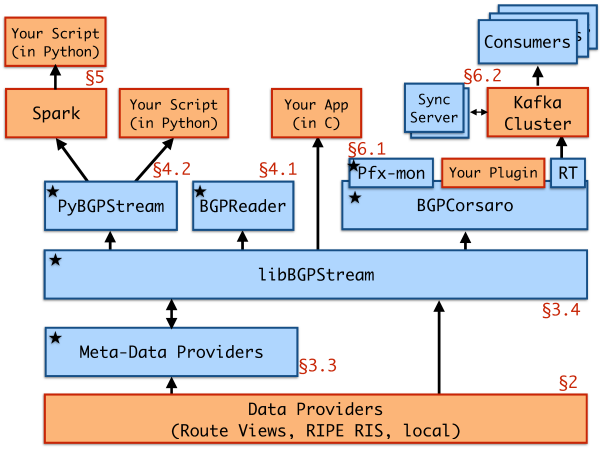


Figure 2: **BGPStream framework overview.** Blue boxes represent components of the framework; those marked with a star are distributed as open source in the current BGPStream release [11]. Orange boxes represent external projects or placeholders. Section numbers mark where each component is discussed in this paper.

3. BGPSTREAM CORE

The BGPStream framework is organized in multiple layers (Figure 2). We discuss the core layers (*meta-data providers* and *libBGPStream*) in this section, whereas we illustrate the upper layers, through case studies, in the remainder of the paper. Meta-data providers serve information about the availability and location of data from *data providers*, (either local or remote) which are data sources external to the BGPStream project.

libBGPStream, the main library of the framework (Section 3.3), provides the following functionalities: *(i)* transparent access to concurrent dumps from multiple collectors, of different collector projects, and of both RIB and Updates; *(ii)* live data processing; *(iii)* data extraction, annotation and error checking; *(iv)* generation of a time-ordered stream of BGP measurement data; *(iv)* an API through which the user can specify and receive a stream.

We distribute BGPStream with the following independent modules: *BGPReader*, a command-line tool that outputs the requested BGP data in ASCII format; *PyBGPStream*, Python bindings to the libBGPStream API; *BGPCorsaro*, a tool that uses a modular plugin architecture to extract statistics or aggregate data that are output at regular time bins.

3.1 Goals and Challenges

We designed the BGPStream framework with the following goals:

– *Efficiently deal with large amounts of distributed BGP data.* In Section 2, we emphasized the importance of performing analyses by taking advantage of a large number of globally distributed VPs. However, dealing with such large amounts of data as well as the distributed and diverse nature (different timing, formats,

etc.) of the VPs pose a series of technical challenges.

– *Offer a time-ordered stream of data from heterogeneous sources.* BGPStream aims to provide a unified sorted stream of data from multiple collectors. Record-level sorting (rather than interleaving dump files) is important in at least two cases: (i) when analyzing long time intervals where time alignment cannot be achieved by buffering the entire input, and (ii) when an input data source provides a continuous stream of data (rather than a discrete dump file), since such a stream cannot be interleaved at the dump file level.

– *Support historical and near-realtime data processing.* We consider two modes of operation: *(i)* **historical** - all the BGP data requested is available before the program starts; *(ii)* **live** - the BGP data requested becomes available while the program is running. In live mode, the BGPStream stack plus the user application, must process data faster than it is generated by VPs/collectors. We minimize BGPStream processing latency, thus maximizing the time available for near-realtime user applications to perform live Internet monitoring and measurements (Sections 4.3 and 6).

Live mode also introduces the problem of sorting records from collectors that may publish data at variable times. This problem involves a trade-off between: (i) size of buffers, (ii) completeness of data available to the application, (iii) latency. Since such a trade-off should be evaluated depending on the specific goals and resources of the user application, we design BGPStream to perform best-effort record interleaving in live mode and we defer to the application the choice of a specific solution (in Section 6.2, we provide a concrete example of such a solution).

– *Target a broad range of applications and users.* Potential applications of BGPStream are both in the field of network monitoring and troubleshooting as well as scientific data analysis. The target user base should not be limited to the availability of high-performance computing and/or cluster infrastructure. The BGPStream framework makes available a set of tools and APIs that suit different applications and development paradigms (e.g., historical data analysis, rapid prototyping, scripting, live monitoring).

– *Scalability.* Since the pervasiveness of BGP VPs is key to monitoring and understanding the Internet infrastructure, the number of VPs supported by collector projects continually grows. In parallel, the technological challenges (e.g., near-realtime detection of sophisticated man-in-the-middle attacks [19, 20]) require solutions of increasing complexity and computational demand. We designed BGPStream to enable deployment in distributed and "Big Data analytics" environments: e.g., Spark's [2] native Python support makes BGPStream usable in such an environment out-of-the-box (Section 6.2).

– *Easily extensible.* Though our solution is designed to work with current standards and the most popular available data sources, we designed a stacked and modular framework, facilitating support for new technologies and data sources. BGPStream is indeed a project under evolution and is part of a coordinated effort with data providers, developers of complementary technologies, and users, to advance the state of the art in BGP monitoring and measurement data analysis [13, 17].

3.2 BGPStream Meta-Data Providers

One of the challenges in analyzing BGP measurement data is identifying and obtaining relevant data. Both RouteViews and RIPE RIS make data available over HTTP, with basic directory-listing style indexes into the data. Identifying the appropriate files for large-scale analysis (across multiple collectors and long time durations) involves either manual browsing and download, or scripting of a crawler tailored to the structure of each project's repository. Downloading the data, may itself take a significant amount of time (e.g., all data collected in 2014 is ≈2TB). Moreover, since both projects continually add new data to their archives as it is collected (Section 2), near-realtime monitoring requires custom scripts to periodically scrape the websites and download new data. BGPStream hides all of these complexities through meta-data providers: components that provide access to information about the files hosted by local or remote data repositories (the *Data Providers*, e.g., the RouteViews and RIPE RIS archives).

We implemented such a meta-data provider as a web service called *BGPStream Broker*, responsible for (i) providing meta-data to libBGPStream, (ii) load balancing, (iii) response windowing for overload protection, (iv) support for live data processing. The Broker continuously scrapes data provider repositories, stores meta-data about new files into an SQL database, and answers HTTP queries to identify the location of files matching a set of parameters. An instance of the Broker is hosted at the San Diego Supercomputer Center at UC San Diego and is queried by default by a libBGPStream installation, allowing BGPStream to be used "out-of-the-box" on any Internet-connected machine.

The Broker stores only meta-data about files available on the official repository, not the files themselves. This approach minimizes the potential for a bottleneck since queries to, and responses from, the Broker are lightweight, with the actual data being served by external data provider archives. This configuration also makes it simple to add support for additional data providers, as well as provide load-balancing and redundancy as the Broker can transparently round-robin amongst multiple mirror servers or adopt more sophisticated policies (e.g., requests sent from UC San Diego machines are normally pointed to campus mirrors).

While the Broker Data Interface is the primary data access interface, we also provide three other interfaces for analysis of local files: *Single file*, *CSV file*, and *SQLite*. The following sections assume that the Broker is used as the Data Interface.

3.3 libBGPStream

3.3.1 Application Programming Interface

The libBGPStream user API provides the essential functions to configure and consume a stream of BGP measurement data and a systematic organization of the BGP information into data structures. The API defines a BGP data stream by the following parameters: *collector projects (e.g., RouteViews, RIPE RIS), list of collectors, dump types* (RIB/Updates), *time interval start* and either *time interval end* or *live mode*. A stream can include dumps of different type and from different collector projects.

On the BGPStream website [12] we provide tutorials with sample code to use the BGPStream API. In general, any program using the libBGPStream C API consists of a stream configuration phase and a stream reading phase: first, the user defines the meta-data filters, then the iteratively requests new records to process from the stream. Code can be converted into a live monitoring process simply by setting the end of the time interval to *-1*.

3.3.2 Interface to Meta-Data and Data Providers

To access data and meta-data from the providers, the library implements a "client pull" model, which enables efficient data retrieval without potential input buffer overflow (i.e., data is only retrieved when the user is ready to process it).

To implement this model, the system iteratively alternates between making meta-data queries to the Broker and accessing and processing the dump files whose URLs are returned by the Broker. When the Broker returns an empty set of dump file URLs, the system signals to the user that the stream has ended. In live mode however, the query mechanism is blocking: if the Broker has no data available, libBGPStream will poll until a response from the Broker points to new data for processing.

3.3.3 Data structures and error checking

libBGPStream processes dump files [9] composed of *MRT records*. While an update message is stored in a single MRT record, RIB dumps require multiple records. The *BGPStream record* structure contains a de-serialized MRT record, as well as an error flag, and additional annotations related to the originating dump (e.g., project and collector names).

To open MRT dumps, we use a version of libBGPdump [54] that we extended to: (i) read remote paths (HTTP and HTTPS), (ii) support reading from multiple files in parallel from a single process, and (iii) signal a corrupted read. libBGPStream uses this signal to mark a record as *not-valid* (*status* field) when the BGP dump file cannot be opened or if the dump is corrupted. libBGPStream also marks records that *begin* or *end* a dump file, allowing users to collate records contained in a single RIB dump.

An MRT record (and therefore a BGPStream record) may group elements of the same type but related to different VPs or prefixes, such as routes to the same prefix from different VPs (in a RIB dump record), or announcements from the same VP, to multiple prefixes, but sharing a common path (in a Updates dump record). To provide access to individual elements, libBGPStream decomposes a record into a set of *BGPStream elem* structures. Table 1 shows the fields that comprise a BGPStream elem. The *AS path* field contains all information present in the underlying BGP message, as specified in RFC 4271 [52], including *AS_SET* and *AS_SEQUENCE* segments. libBGPStream also provides convenience functions for easily iterating over segments in an AS path, accessing fields within a segment, and converting paths and segments to strings (using the same format as bgpdump). We do not currently expose all the BGP attributes contained in a MRT record in the BGPStream elem; we will implement the remaining attributes in a future release. The *old state* and *new state* fields refer to elems from RIPE RIS VPs. Each RIPE RIS collector maintains, for each VP, a Finite State Machine (FSM) for the status of the BGP session with the VP, we store the previous and current state of the FSM.

Table 1: **BGPStream elem fields.**

Field	Type	Function
type	enum	route from a RIB dump, announcement, withdrawal, or state message
time	long	timestamp of MRT record
peer address	struct	IP address of the VP
peer ASN	long	AS number of the VP
prefix*	struct	IP prefix
next hop*	struct	IP address of the next hop
AS path*	struct	AS path
community*	struct	community attribute
old state*	enum	FSM state (before the change)
new state*	enum	FSM state (after the change)

* denotes a field conditionally populated based on type

3.3.4 Generating a sorted stream

libBGPStream generates a stream of records sorted by the timestamps of the MRT records they encapsulate. Collectors write records in dump files with monotonically increasing timestamps. However, additional sorting is necessary when the stream is configured to include MRT records stored in files with overlapping time intervals[2], which occurs in two cases: *(i)* when reading dumps from more than one collector (inter-collector sorting); *(ii)* when a stream is configured to include both RIB and Updates dumps (intra-collector

[2]We define the time interval associated with a dump file as the time range covered by the timestamps of its records.

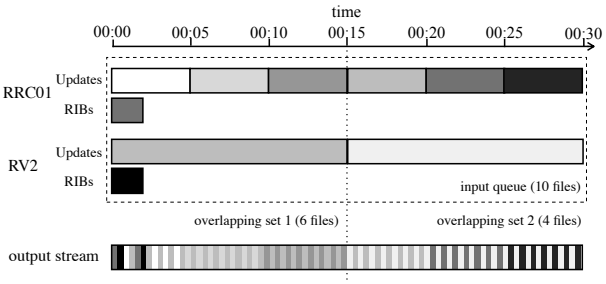


Figure 3: **Intra- and inter-collector sorting in libBGPStream.** An example showing how RIB and Updates dumps generated by a RIPE RIS collector (RRC01) and a RouteViews collector (RV2) are interleaved into a sorted stream. The 30 minutes (10 files) of BGP data are first separated into two disjoint sets (of 6 and 4 files) based on overlapping file time intervals. Then a multi-way merge is applied separately to the two sets, yielding the stream depicted at the bottom.

sorting). Since each file can be seen as an ordered queue of records, in practice libBGPStream performs a *multi-way merge* [38] on such queues.

Given the current number of collectors in RouteViews and RIS (about 30), and that the broker returns in each response a set of dump file URLs (*dump file set*) spanning up to 2 hours of data, the number of files to read can be up to ≈500. The computational cost of the multi-way merging is proportional to the number of queues (files) considered. We therefore break the dump file set in disjoint subsets, ensuring that we place files with overlapping time intervals in the same subset, and apply multi-way merge to each. This problem is exacerbated by the fact that the duration of Updates dumps vary between projects (e.g., RouteViews 15 min, RIPE RIS 5 min). We minimize the number of files per subset by iteratively applying the following steps until we process all files: *(1)* initialize a new subset with the oldest file in the set; *(2)* recursively add files with time intervals overlapping with at least one file already in the subset; *(3)* remove from the set the files in the subset. The subsets we obtain this way typically contain up to ≈150 files. For each of them, we perform the multi-way merge: libBGPStream simultaneously opens all the files in the set and iteratively *(i)* extracts the oldest MRT record from such files, and *(ii)* uses the MRT record to populate a BGPStream record (Figure 3).

We empirically tested the cost of our sorting algorithm by using libBGPStream to process one day of Update and RIB dumps from all collectors of RouteViews and RIPE RIS, and confirmed that the cost of sorting is negligible compared to the cost of actually reading records from the dump files.

4. ANALYSIS OF SPECIFIC EVENTS AND PHENOMENA

While users can write code that directly uses the BGPStream C API, we provide solutions that allow complex measurement experiments to be expressed with little to no code. We show how BGPReader and PyBGStream can rapidly address a variety of research tasks.

4.1 ASCII command-line tool

BGPReader is a tool to output in ASCII format the BGPStream records and elems matching a set of filters given via command-line options. This tool is meant to support exploratory or ad-hoc analysis using command line and scripting tools for parsing ASCII data. BGPReader can be thought of as a drop-in replacement of the analogous bgpdump tool (a command line option sets bgpdump output format), which is widely used by researchers and practitioners. However, BGPReader adds features such as the support to read data from multiple files, collectors, and projects in a single process, the ability to work in live mode and various filters.

For example, the following command line will dump on *stdout* a (sorted) stream of lines, each representing BGP updates from all the RouteViews and RIS collectors which are related to subprefixes of *192/8* and observed since May 12th 2016: `bgpreader -w 1463011200 -t updates -k 192.0.0.0/8`. The command will run indefinitely, outputting new data as it is made available by the data sources.

4.2 Python bindings

PyBGPStream is a Python package that exports all the functions and data structures provided by the libBGPStream C API. We bind directly to the C API instead of implementing the BGPStream functions in Python, in order to leverage both the flexibility of the Python language (and the large set of libraries and packages available) as well as the performance of the underlying C library. Even if an application implemented in Python using PyBGPStream would not achieve the same performance as an equivalent C implementation, PyBGPStream is an effective solution for: rapid prototyping, implementing programs that are not computationally demanding, or programs that are meant to be run offline (i.e., there are no time constraints associated with a live stream of data).

In Listing 1, we show a practical example related to a research topic commonly studied in literature: the AS path inflation [33]. The problem consists in quantifying the extent to which routing policies inflate the AS paths (i.e., how many AS paths are longer than the shortest path between two ASes due to the adoption of routing policies), and it has practical implications, as the phenomenon directly correlates to the increase in BGP convergence time [40]. In less than 30 lines of code, the program compares the AS-path length observed in a set of BGP RIB dumps and the corresponding shortest path computed on a simple undirected graph built using the AS adjacencies observed in the AS paths. The program reads the 8am RIB dumps provided by all RIS and RouteViews collectors on August 1st 2015, and ex-

Listing 1 Calculate AS path inflation in **≈30 lines of code**. Shows a fully-functional Python script that processes the 8am RIB dumps from all RouteViews and RIPE RIS collectors on August 1 2015, and compares the AS-path length observed in a set of BGP RIB dumps with the corresponding shortest path computed on a simple undirected graph built using the same BGP data.

```
from _pybgpstream import BGPStream, BGPRecord, BGPElem
from collections import defaultdict
from itertools import groupby
import networkx as nx

# create and configure new BGPStream instance
stream = BGPStream()
rec = BGPRecord()
# request RIB data from Aug 1 2015 7:50am -> 8:10am (UTC)
stream.add_filter('record-type', 'ribs')
stream.add_interval_filter(1438415400, 1438416600)
stream.start()

# create datastructures for the undirected graph and path lengths
as_graph = nx.Graph()
bgp_lens = defaultdict(lambda: defaultdict(lambda: None))

# consume records from the stream
while(stream.get_next_record(rec)):
    # process all elements of each record
    elem = rec.get_next_elem()
    while(elem):
        monitor = str(elem.peer_asn)
        # split the AS path into segments
        hops = [k for k, g in groupby(elem.fields['as-path'].split(" "))]
        # sanitization: ignore local routes
        if len(hops) > 1 and hops[0] == monitor:
            origin = hops[-1]
            # add all edges to the NetworkX graph
            for i in range(0,len(hops)-1):
                as_graph.add_edge(hops[i],hops[i+1])
            # how long this path is (for comparison to shortest path)
            bgp_lens[monitor][origin] = \
                min(filter(bool,[bgp_lens[monitor][origin],len(hops)]))
        elem = rec.get_next_elem()
# compare actual BGP path lengths to computed shortest path
for monitor in bgp_lens:
    for origin in bgp_lens[monitor]:
        nxlen = len(nx.shortest_path(as_graph, monitor, origin))
        print monitor, origin, bgp_lens[monitor][origin], nxlen
```

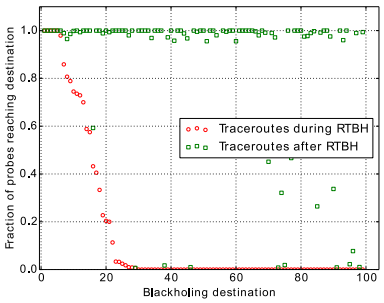
tracts the minimum AS-path length observed between a VP and each origin AS. While reading the RIB dumps, the program also maintains the AS adjacencies observed in the AS path. We then use the NetworkX package [47] to build a simple undirected graph (i.e., a graph with no loops, where links are not directed) and we compute the shortest path between the same <VP, origin> AS pairs observed in the RIB dumps. In this example, we compare path lengths of 10M unique <VP, origin> AS pairs and find that, in more than 30% of cases, inflation of the path between the VP's AS and the origin AS accounts for 1 to 11 hops. Compared to the study of Gao and Wang, who analyzed RouteViews data from year 2000 and 2001 [33], these numbers show more inflated paths (>30% instead of >20%) and a consistent number of max additional hops (11 instead of 10).

In conclusion, this case study shows that BGPStream simplifies the task of analyzing heterogeneous BGP data, especially if we want such analysis to be systematically applied to different datasets and repeatable. To perform exactly the same experiment described here without using BGPStream, a researcher would have to manually identify and download all the data and write a parser for bgpdump ASCII output (besides writing the same core analysis logic). In addition, if a researcher wants to perform the same type of analysis on different sets of data (e.g., time window, subset of collectors, data collection projects) they would need to repeat the manual work of identifying and downloading all the files needed or, more realistically, develop a configurable crawler supporting the file hierarchies and naming conventions of different collection projects. Moreover, if another research task requires code to be aware of data collection time, type (RIB and Updates dumps) and provenance (collector, project), a researcher would need to build a data indexing system accessible to analysis code. While such efforts are doable, first they represent an added cost that greatly outweighs the cost of writing the core analysis code and deviates focus from the specific research task. Second, in a research context, similar efforts typically result in a mix of ad-hoc code and scripts that make sharing for reproducibility purposes improbable. Finally, since a script using PyBGPStream embeds the full definition of the input data used for an experiment, it further fosters reproducibility of experimental results.

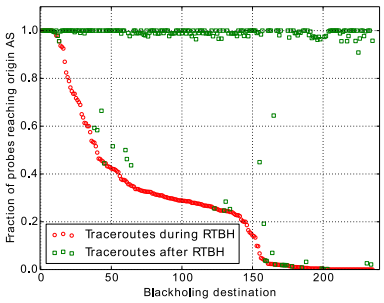
4.3 Timely Additional Measurements

In this section, we show how BGPStream can be combined with RIPE Atlas [55], a large-scale distributed infrastructure for active measurements, to enable timely analysis of customer-triggered BGP policies.

To mitigate the collateral damage of a DoS attack, a customer may use the Remotely Triggered Black-Holing (RTBH) technique to request its transit providers or peers to divert all the traffic towards its targeted IP addresses to a *null interface*, which drops all the incoming traffic [39]. Providers who support RTBH define a BGP community [15] [59] that can be used by their customers to signal IP ranges to be black-holed. Since BGP communities lack standardization, the black-holing communities can differ among different providers [26], therefore multi-homed customers may need to set multiple black-holing communities to request black-holing from more than one of their providers. RTBH is effective on minimizing the collateral damage of DoS attacks, at the expense of taking the target completely offline. To limit the number of hosts adversely affected by black-holing, providers often restrict black-holing only to /32 prefixes, although shorter prefixes may also be allowed depending on each provider's policy. The lack of standardization regarding RTBH policies means that the impact of RTBH can be difficult to predict. Dietzel et. al. have studied the use of RTBH from the perspective of a large European IXP [25]. In this case study, we



(a) Fraction of traceroute queries that reach each black-holed destination.



(b) Fraction of traceroute queries per black-holed destination that reach each origin AS.

Figure 4: Two metrics showing a pronounced difference in the data-plane reachability of black-holed destinations during (red) and after RTBH (green). For each destination we execute traceroutes from 50-100 Atlas probes (depending on the connectivity of the origin AS), which we repeat after blackholing is withdrawn. The results are ordered based on the values of each metric during RTBH.

combine data-plane and control-plane measurements to demonstrate how we can gain a better understanding of how black-holing is implemented and its effects. Our purpose is to illustrate how BGPStream filters and live-mode streams facilitate complicated measurements that otherwise would require enormous instrumentation efforts, rather than providing a complete study of RTBH.

We identify as an RTBH request any triple of (collector, VP, prefix) that is tagged with at least one black-holing community from a list we compiled by parsing the IRR records and technical support websites for 30 ASes (13 Tier-1 providers, 12 multinational ISPs, and 5 academic networks). We respectively mark the *start* of an RTBH request when we first observe a BGP update with a black-holing community attached on a prefix that was previously announced without such a community, and the *end* when such prefix is re-advertised without it or explicitly withdrawn.

We executed our RTBH measurements between 20-29 April 2016 by continuously listening to BGP updates from the *route-views2* and *RRC12* collectors, for IPv4 prefix announcements tagged with black-holing communities. Almost 80% of the RTBH requests we detected have a duration of less than a day, while 20% have a duration of less than 40 minutes. These observations are consistent with previous studies on DoS attack duration [6,25]. Therefore, it is important to minimize the delay between the application of black-holing communities and the detection time, in order to avoid missing the time window during which we can execute traceroute measurements toward the black-holed prefixes. To minimize latency between BGP and traceroute measurements, we utilize two BGPStream streams (within the same Python script) running in live mode to collect BGP updates. We apply community-based filters to the first stream so that it only yields prefix announcements tagged with at least one black-holing community. Whenever we observe a RTBH request from this stream, we add a filter for the black-holed prefix to the second stream to capture explicit or implicit withdrawals. Using two streams in this manner provides a clear separation of concerns, simplifying the logic in our Python script. That is, one stream triggers investigation of a prefix, whereas the other (possibly) triggers the completion of investigation.

Upon detecting the start of an RTBH request we orchestrate a set of *paris* ICMP traceroutes towards a random IP address in the corresponding prefix. We select currently-active RIPE Atlas probes from: (i) the visible AS neighbors of the origin AS, (ii) ASes that are co-located in the same IXPs as the origin AS, (iii) the same country of the target IP (to account for potentially invisible peripheral peering inter-connections). Our measurements are timely in most of the cases: we are able to probe over 95% and 90% of the black-holed prefixes, respectively for updates collected from RIPE RIS and RouteViews, before the RTBH is switched off. We also repeat the same traceroutes as we detect the end of the RTBH request.

In total, we discovered 482 black-holed prefixes, originated by 67 different ASes. 398 of the black-holed prefixes had a length longer than /24, 397 of which had a length of /32 (single hosts). Contrary to the best practices that recommend the suppression of black-holed prefix advertisements [16, 39] or prefixes that are too specific [27], during the short period of our experiment we observed a non-trivial number of black-holed prefixes that propagated beyond the AS that defined the balck-holing communities. Namely, the corresponding ASes applied neither the egress filter for black-holed prefixes, nor the egress filter for too specific prefixes. Past works found that prefixes longer than /24 are visible to 20% – 30% of the monitors at the BGP collectors [4,10]. In Section 5 we briefly analyze the propagation of BGP communities as it is visible from BGP collectors. However, the control-plane propagation of the black-holed prefixes beyond the network that applies the black-holing has not been analyzed before. From

our measurement results, we remove prefixes for which we could not obtain traceroutes from the same set of Atlas probes between the two measurements (due to fluctuations in the availability of the probes), obtaining 253 prefixes that we briefly investigate (Figure 4).

Figure 4a shows the fraction of traceroutes that reach each destination. In this graph we do not include destinations that traceroutes do not reach after the RTBH, resulting in 100 destinations examined: after the RTBH 83% destinations are reached by at least 95% of the traceroutes, whereas during the RTBH 77% of the destinations are reached by less than 5% of the traceroutes and 73% are never reached by any probe. These numbers clearly show a change in data-plane reachability. Interestingly, during RTBH, 13% of the destinations are only partially reachable (between 20% and 80% reachability). For these destinations, we manually verified that traceroutes from customers or peers of the origin AS could still reach the black-holed destination, while ASes in the upstream path failed.

Although these results indicate a change in data-plane reachability during the RTBH events, the DoS attack itself may be the cause of traceroutes not reaching a host normally responding. However, if RTBH is in place the traffic is supposed to be dropped at the border routers [39, 61]. To dig deeper, in Figure 4b (where we consider all the 253 prefixes) we look at reachability at the level of the origin AS instead of the end host. On one hand, we find that the majority of the destinations (190) experience traceroutes that frequently fail (i.e., 40% reachability or less) to reach the origin AS. On the other hand, the vast majority of destinations show full reachability of the originAS after the RTBH. This preliminary finding is consistent with the expected behavior when RTBH is employed, and we plan to investigate it further in future work.

In conclusion, this case study shows that we are able to easily instrument combined passive control-plane and active data-plane measurements to capture and investigate transient routing policies. Without the capability to stream and filter BGP updates in near-realtime, we would be unable to capture the data-plane paths of the short-lived black-holing events. The alternative to BGPStream (or a system that implements the same features) would have been to continuously run traceroute scans against the entire address space which is impractical given the immense resource requirements of such an exhaustive probing.

5. ANALYSIS OF MASSIVE DATASETS

In this section, we showcase simple case studies to demonstrate that BGPStream's Python bindings are readily usable in a Big Data workflow. We deploy PyBGPStream scripts in an Apache Spark [2] environment running on a 15-node cluster (240 CPUs and 960GB of RAM) to extract statistics of BGP across the last 15 years. We make these scripts available at [14] as a starting point for other researchers to use PyBGPStream with Spark for their own analyses. In addition, our examination highlights how certain features of the BGP eco-system (e.g., average size of the routing table) appear different depending on the data sources and data types picked from the heterogeneous BGP measurement infrastructure currently available to researchers, thus providing a reference for future research.

In all our analyses, we processed the midnight RIB dumps of the 15th day of each month from January 2001 to January 2016: more than 3000 RIB dumps, totalling approximately 44 billion BGP elems. The running times of the various analyses range between ≈1 and 24 hours. All our Python scripts share a common structure: *(i)* we build a list of data partitions splitting the data by time range and BGP collector and instruct Spark to create a Resilient Distributed Dataset (a data structure split across many nodes)[3]; *(ii)* we map a Python function to execute for every element in the list; this function represents the core of the BGPStream routines for data extraction; e.g., it creates the stream (defining filters etc.) and it executes nested *while* loops – per BGP record and per BGP elem – such as the one in Listing 1; this operation results in the creation of as many streams as list elements; *(iii)* we specify three independent reduction operations: per VP, per collector, overall. We also provide a *hello-world* template script that follows this pattern [14].

In our first analysis, we study the growth of the IPv4 routing table in BGP speakers over time (calculated as the number of unique prefixes in the Adj-RIB-out of each VP). There are three observations in this analysis useful as future reference for similar studies: *(i)* partial-feed VPs, i.e., those showing significantly smaller Adj-RIB-outs, are numerous and significantly skew the distribution; Figure 5a shows a heatmap of data from 2,296 VPs, with warmer colors representing a higher concentration of points from different VPs; only 710 out of 2,296 VPs are within 20 percentage points of the maximum at each time bin (we adopt this definition of full-feed VP in the following); *(ii)* two collectors (RouteViews *kixp* and *soxrs*) do not have a single full-feed peer, thus may not provide enough information for most analyses; *(iii)* we find that both the RouteViews and RIPE RIS repositories occasionally miss RIB dumps (34 per year on average) on midnight of the 1st day of the month (thus we perform our analyses with data from the 15th day of the month). In this analysis, we also compute, at each level of aggregation (VP, collector, overall), the number of unique prefixes and ASes observed, which we use to normalize data in the other analyses.

Figure 5b shows the results of analysis in which identified MOAS (Multi Origin AS) prefixes [64]. Study and detection of MOAS prefixes is relevant to many problems [36], including the detection of BGP hijacking

[3]We also specify the number of slices, typically 2-3 times the number of cores in the cluster.

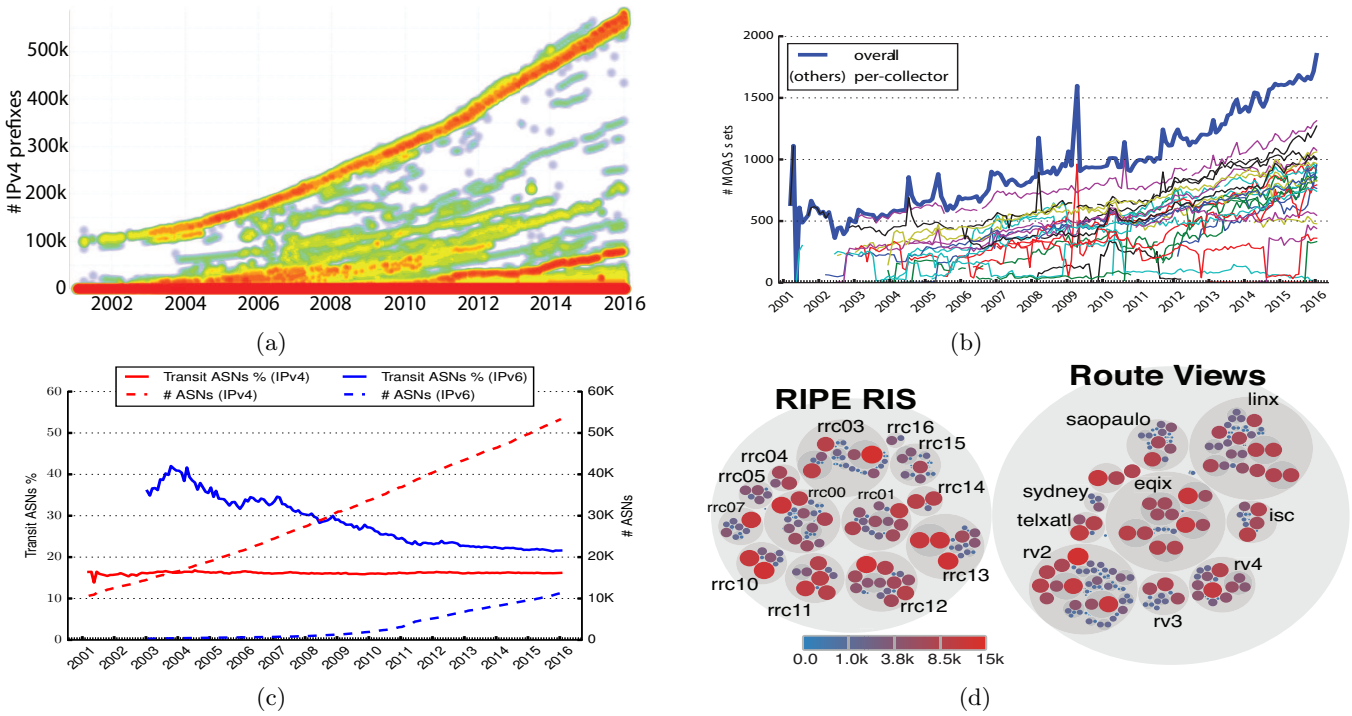


Figure 5: **Results of historical analysis using PyBGPStream and Apache Spark.** *(a)* heatmap depicting the growth of the IPv4 routing table in VPs over time. The y axis shows the number of prefixes in the Adj-RIB-out of VPs; warmer colors represent a higher concentration of points. *(b)* number of unique MOAS sets (y axis) over time, aggregated into overall (top blue line) and per-collector (other lines). *(c)* absolute number of ASNs (dashed lines) and percentage of those ASNs which are classified as transit – i.e., appearing in the middle of an AS path – (solid lines), for both IPv4 (red lines) and IPv6 (blue lines). *(d)* community diversity as observed by VPs (January 2016). VPs are depicted as circles (inner colored circles) with diameter and color proportional to the number of distinct AS identifiers (inferred from the two most-significant bytes of the community value) in the BGP communities they observe. The aggregated data (by collector and by data collection project) are depicted as grey circles, and highlight which collectors observe a more heterogeneous set of BGP communities.

activity [20]. Figure 5b plots the number of unique sets of ASes (*MOAS sets* in the following) contributing to MOAS prefixes aggregated into overall (top blue line) and per-collector (other lines). Besides the slow growth in observable MOAS sets over time, this graph highlights that to obtain a better view of MOAS prefixes, it is important to analyze data from as many collectors as are available: the number of MOAS sets identified in the *overall* aggregation is always significantly larger than the maximum number identified by a single collector.

We then calculated the number of transit ASes (ASes appearing in the middle of an AS path) observed for both IPv4 and IPv6. Figure 5c shows that for IPv4, despite the nearly-linear growth in the number of ASes, the fraction of transit ASes over time has been constant! For IPv6 in contrast, overall there has been a constant decay in the fraction of transit ASes (edge growing faster than transit). However, around 2012, this decay slowed considerably, while the total number of IPv6 ASes kept a fast rate: the IPv6 graph is growing fast while its edge and transit portions recently started growing at similar paces! (Approaching the property we observed in the IPv4 graph over the last 15 years.) As of January 2016, however, the fraction of transit ASes is much larger in IPv6 (21% vs 16%), reflecting a smaller adoption of IPv6 at the edge.

In the final analysis we conducted with Spark, we investigated how BGP communities propagate and are visible via the RouteViews and RIPE RIS measurement infrastructures. BGP communities can be used to study several relevant Internet phenomena, such as complex AS relationships [34], traffic engineering policies [51], DDos mitication (Section 4.3). We collected unique communities appearing in IPv4 paths and we found that the number of observable communities over time increased from ≈800 (January 2001) to ≈40,000 (January 2016). In the rest of this section we focus on the most recent data (January 2016). By counting only the AS identifier portion of each community (which typically refers to the AS targeted by or generating the community), we observed approximately 4,000 ASes using communities. We observe communities only through ≈83% of the VPs, showing that many BGP speakers strip out communities from AS paths before propagating them. By observing the full paths, we find that at least 1,000 ASes propagate BGP communities (out of the more than 8,000 transit ASes found in the previous

analysis). In practice, since the number of communities a VP observes depends on the filtering by the ASes in its vicinity, analysis requiring either diversity of BGP communities or communities from a specific AS requires a careful choice of VPs/collectors. Figure 5d[4], shows the VPs as circles (inner colored circles) with diameter proportional to the number of distinct AS identifiers (inferred from the two most-significant bytes of the community value) in the BGP communities they observe. The aggregated data (by collector and by data collection project) are depicted as grey circles, and highlight which collectors observe a more heterogeneous set of BGP communities: RouteViews collectors *route-views2* (3,624), *linx* (3,262), *route-views4* (3,236), and RIPE RIS collectors *rrc04* (2,979), *rrc01* (2,947), and *rrc12* (2,886). We selected the two collectors used for the analysis in Section 4.3 based on these data.

Performing analyses such as those discussed in this section without using BGPStream requires considering scalability issues (besides the efforts described in Section 4.2: crawling, data indexing, ASCII output parsing). For example, the amount of data needed for large longitudinal analyses may preclude a-priori download. In this case, a researcher would need to develop a system to dynamically download a moving window of data for consumption by analysis code. However, such a solution will turn storage into a potential bottleneck, since the size of the window limits the number of processing units that can run in parallel. A better solution would instead enable processing scripts to download data on demand, which is close (but still suboptimal) to the functionality provided by libBGPStream (which does not download the file to disk but streams it to the script directly from the HTTP connection). Another benefit of such functionality in a cluster-computing context is that it reduces the overhead of data locality optimization, since it implicitly co-locates each data block with the appropriate processor.

6. CONTINUOUS MONITORING

6.1 Lightweight monitoring: BGPCorsaro

BGPCorsaro is a tool to continuously extract derived data from a BGP stream in regular time bins. Its architecture is based on a pipeline of plugins, which continuously process BGPStream records. Plugins can be either:

- Stateless: e.g., performing classification and tagging of BGP records; plugins following in the pipeline can use such tags to inform their processing.
- Stateful: e.g., extracting statistics or aggregating data that are output at the end of each time bin. Since libBGPStream provides a sorted stream of records, BGPCorsaro can easily recognize the end of a time bin even when processing data from multiple collectors.

[4]An interactive, high-resolution version of this graph, as well as the equivalent for IPv6, are available at [14].

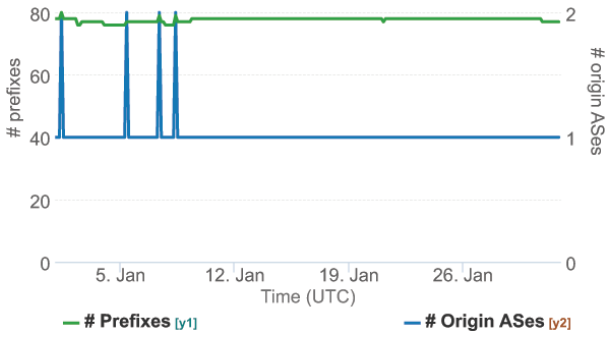


Figure 6: **Monitoring of GARR (AS137) IP space using the pfxmonitor plugin.** The green line is the number of unique prefixes announced over time. The blue line is the number of unique origin ASes that announce them: the spikes identify four hijacking events in which AS 198596 announces part of the IP space belonging to AS137.

Both the core and the plugins of BGPCorsaro are written in C in order to support high-speed analysis of historical or live data streams. In Section 6.2, we describe a deployment of BGPCorsaro that runs 24/7 as a part of our global Internet monitoring infrastructure.

As a sample plugin, we describe a stateful plugin that monitors prefixes overlapping with a given set of IP address ranges. For each BGPStream record, the plugin: (1) selects only the RIB and Updates dump records related to prefixes that overlap with the given IP address ranges. (2) tracks, for each <prefix, VP> pair, the ASN that originated the route to the prefix. At the end of each time bin, the plugin outputs the timestamp of the current bin, the number of unique prefixes identified and, the number of unique origin ASNs observed by all the VPs.

We use a BGP hijacking event reported by Dyn Research, the hijacking of Italian Academic and Research Network (GARR) prefixes on January 7th 2015 [45], to demonstrate this plugin. We configured the plugin to process data from all available RouteViews and RIPE RIS collectors for January 2015, setting the time bin size to 5 minutes, and providing as input to the plugin the IP ranges covered by the 78 prefixes originated by AS137 (GARR) as observed on January 1st, 2015. Figure 6 shows a graphical representation of the two time-series generated by the plugin: the number of unique announced prefixes (in green) and number of unique origin ASNs (in blue). While a small oscillation of the number of prefixes announced is expected (as prefixes can be announced as aggregated or de-aggregated), in 4 cases the number of unique announcing ASes shifts from 1 to 2, for about 1 hour. Through manual analysis, we found that during these spikes a portion of GARR's IP space (specifically, 7 /24 prefixes) was also announced by TehnoGrup (AS 198596), a Romanian AS that appears to have no relationship with GARR. The report

by Dyn Research describes a single attack on January 7th. However, given the similar nature of the other three events visible in the graph (1st, 5th and 8th of January), the plugin output suggests that three additional attacks occurred. Although this approach cannot detect all types of hijacking attacks, it is still a valid method to identify suspicious events and serves to demonstrate how users can leverage the capabilities of BGPCorsaro by writing plugins specific to their application.

6.2 Monitoring the Global Internet

In this section, we present a distributed architecture built on top of BGPStream and leveraging Apache Kafka [1] (a distributed messaging system) to perform continuous global BGP monitoring. Our goal is two-fold: we demonstrate how BGPStream enables and simplifies developing complex global monitoring infrastructure and we present our architectural solutions to challenges that arise in this context.

To frame context and motivation for developing such complex architectures, let us consider two sample applications, our "Internet Outages: Detection and Analysis" (IODA) [23] and "Hijacks" [20] research projects. In IODA we monitor the Internet 24/7 to detect and characterize phenomena of macroscopic connectivity disruption [21] [22]. In the case of BGP, our objective is to understand whether a set of prefixes (e.g., that share the same geographical region, or the same origin AS) are globally reachable or not. Information from a single VP is not sufficient to verify the occurrence of an outage, in fact, a prefix may be not reachable from the VP because of a local routing failure. On the other hand, if several VPs, topologically and geographically dispersed, simultaneously lose visibility of a prefix, then the prefix itself is likely undergoing an outage. In Hijacks, we are interested in detecting and analyzing BGP-based traffic hijacking. Since most common hijacks manifest as two or more ASes announcing exactly the same prefix, or a portion of the same address space at the same time, detecting them requires comparing the prefix reachability information as observed from multiple VPs.

In order to detect these events in a timely fashion, we need to maintain a global (i.e., for each and every VP) view of BGP reachability information updated with fine time granularity (e.g., few minutes). Such a continuously updated global view can be useful in many other applications, such as tracking AS paths containing a particular AS, verifying the occurrence of a route leak, spotting new (suspicious) AS links appearing in the AS-graph, etc.

We sketch our proposed architecture in Figure 7: multiple BGPCorsaro process data (one instance per collector, in order to distribute the computation across multiple CPUs/hosts), their output is stored into an Apache Kafka cluster and further processed by applications (*consumers*) based on meta-data generated by *synchronization servers*. In the following sections, we

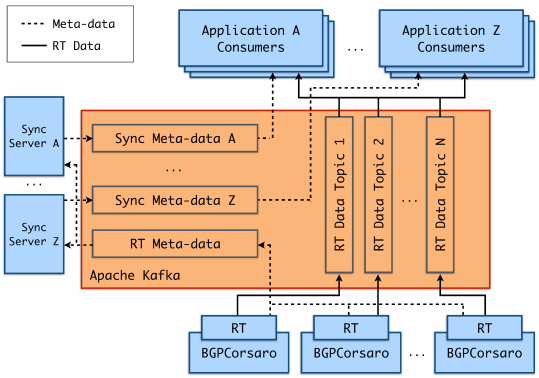


Figure 7: **Distributed framework for live monitoring**. For each collector, we run an instance of BGPCorsaro with the RT plugin which reconstructs the observable LocRIB of all of the collector's VPs. At the end of each time bin (e.g., 1 minute) each BGPCorsaro publishes diffs to a Kafka cluster. Per-application sync servers then align data from multiple collectors and signal consumers to start processing.

describe the main components of this architecture and which challenges they address: Section 6.2.1 explains how we efficiently and accurately reconstruct the observable LocRIB of each VP; Section 6.2.2 illustrates our solution to reduce the amount of data we store and later process with the consumers; Section 6.2.3 shows how we solve the problem of supporting different synchronization mechanisms based on the application requirements; finally, in Section 6.2.4 we provide an example of applications implemented as a consumer.

6.2.1 *Reconstructing VPs routing tables*

RIB dumps are typically available every 2 or 8 hours. Our goal is to reconstruct snapshots of the observable LocRIB (herein referred to as the *routing table*) of each VP with a granularity of 1 or few minutes. For this purpose, we developed a BGPCorsaro plugin, called *routing-tables (RT)*. The RT plugin uses a RIB dump as a starting reference and then relies on the Updates dumps to reconstruct the evolution of the routing table, using subsequent RIB dumps for sanity checking and correction. However, since this is an inference process based on distributed collection of heterogeneous measurement data, multiple things can go wrong: BGP sessions going down, corrupted data, dump files published out of order, etc. We address this problem by maintaining a finite state machine and data structures that model the state of the VP, its routing table, and our confidence that the modeled data is accurate. In particular, we deal with the following four special events: ***E1.*** We ignore *all* records of a RIB dump if libBGPStream marks at least one of its records as corrupted. ***E2.*** Since records from a single RIB dump have timestamps often spanning several minutes *and* RIB and Update dumps may be published out of order, it is possible for the plugin to receive a RIB dump with some records that are

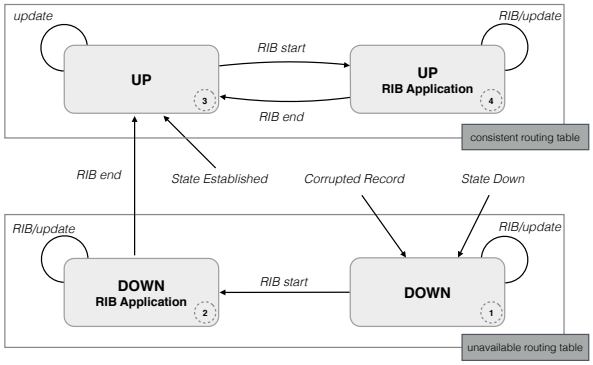


Figure 8: **Finite State Machine (FSM) for reconstructing VP routing table.** The two macro states (*consistent routing table* and *unavailable routing table* represent the (un)availability of a consistent routing table. The FSM starts in the *down* state, then it usually moves to *down-RIB-Application*, and, for the vast majority of time, it oscillates between *up* and *up-RIB-Application*.

older than the latest Update records applied by the plugin. To cope with this issue, we check each individual record of a RIB dump and only apply information from the record if the timestamp of the record is more recent than the timestamp of information already applied by the plugin. ***E3.*** Upon receiving a corrupted Updates dump record we stop applying Updates and wait for the next RIB dump. ***E4.*** We force state transitions upon receiving certain VP state messages (e.g., receipt of a state message with the Established code [52] triggers a transition to the UP state).

We save state and routing table information in a multi-dimensional hash table, which can be seen as a matrix with prefixes as rows and VPs as columns. Each cell contains the *reachability-attributes* for the prefix (e.g., the AS path), the *timestamp* of when the cell was last modified by an Updates dump record, and a *A/W* flag that indicates whether such operation was an announcement or a withdrawal. In addition, for each cell, the RT plugin uses a *shadow cell* to temporarily store records from a new RIB dump until it receives its last record: if none of the RIB dump records are corrupted (***E1***), we replace the content of the main cell with the content of the shadow cell unless the timestamp of the RIB record is older than the cell's last modification time (***E2***).

Figure 8 depicts the process of maintaining a VP routing table as a finite state machine that models the state of the VP. When the plugin starts, the VP's routing table is unavailable and the VP is in state *down* (1). When a new RIB dump starts, the VP's state moves to *down-RIB-application* state (2). During this phase, the plugin populates the shadow cells with the information received from the RIB dump records and the main cells with Updates dump records. The VP's state becomes *up* (3) once the entire RIB dump is received; when in this state the routing table is determined to be an accurate representation of the VP's routing table. Each new announcement or withdrawal record triggers modification of the main cell, whereas if a new RIB dump starts, the VP's state transitions to *up-RIB-application* (4), a state similar to (3) but whereby the RIB dump records modify the shadow information of the cells. Once the RIB ends, the shadow and main cells are merged (as described previously) and the VP transitions to state (3) again. In addition, a corrupted Updates dump record forces the state to be *down* (***E3***). Reception of an Updates dump record carrying a state message[5] with the Established code [52] moves the VP's state to *up* (***E4***), whereas reception of any other state message indicates that the connection between the VP and the collector is not established, and therefore, the VP is considered *down* (***E4***).

To evaluate the accuracy of our approach, we periodically compare the information in the current and shadow cell. RIS and RouteViews error probabilities – defined as the number of mismatching prefixes over the sum of all VPs' prefixes – calculated over 12 months across 31 collectors, are 10^{-8} and 10^{-5} respectively. We find that mismatches are usually caused by unresponsive VPs for which we do not have state messages (e.g., RouteViews), or by a collector not applying all incoming update messages before starting its RIB dump (but applying them afterwards, even if they have been already assigned a timestamp).

6.2.2 IO routines: diffs, (de)serialization, Kafka

At the end of each time bin, the RT plugin transmits the reconstructed routing table of each VP to a Kafka cluster. However, in order to reduce the volume of data to be stored and later processed by the consumers, we developed routines that allow the RT plugin to compute the difference between the routing table generated at the previous time bin and the current one and transmit only the changed portions (which we call *diff cells*). Consumers use complementary routines to retrieve the data from Kafka and reconstruct a full routing table by applying diffs to the previously stored version. The resulting data structure marks the updated portions of the routing table, allowing a consumer to limit its analysis to only these data. We periodically (e.g., 1 hour) also store entire (non-diff) routing tables in the Kafka cluster that applications can use for synchronizing in order to receive future diffs.

Figure 9 highlights the advantage (in terms of number of processed BGP elems) of processing only diffs between routing tables instead of processing every update

[5]Each RIPE collector maintains, for each VP, a finite state machine for the status of the BGP session with the VP and dump specific messages when state transitions occur. RouteViews collectors do not dump such state messages, hence the plugin may maintain a stale routing table for a VP that is actually down. To mitigate this problem, we also declare a VP down if none of its routes are present in the latest RIB dump.

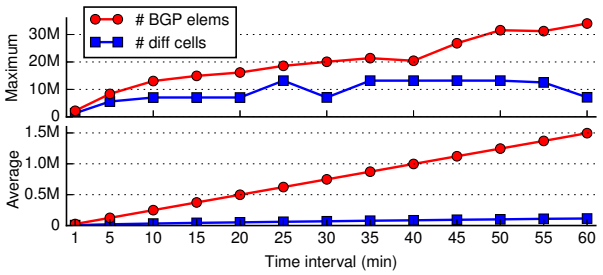


Figure 9: **RT diffs vs. BGP elems.** Results from running the RT plugin on data from *route-views2* for the month of March 2016: average and maximum number (bottom and top graphs respectively) of BGP elems (red circles) and diff cells published by the RT plugin (blue squares) in each time bin.

message. We run the RT plugin on data from *route-views2* for the month of March 2016: in the graph, the red circles show the average (bottom) and maximum (top) number of BGP elems extracted from BGP update messages in each time bin, whereas the blue squares show the number of *diff cells* between consecutive routing tables. When the time bin is 1 minute, there are on average more than 3 times fewer diff cells than BGP elems, indicating that there is redundancy in update messages even at such short time scales. As the size of the time bin increases, the reduction factor also increases, at the expense of time granularity; a time bin of 1 hour yields ≈13 times fewer diff cells than BGP elems. Also, the maxima show that by processing diffs, consumers are more resilient to bursts of updates (e.g., as a result of prefixes flapping).

6.2.3 *Data synchronization*

Different collectors, and in general different data sources, provide data with variable delay. Performing data synchronization requires a trade-off between latency, amount of data available at processing time, and memory footprint. The optimal point in such a trade-off depends on the specific application goals and requirements. A monitoring application may require data from all (or a given fraction of) available sources for the current time bin regardless of latency. Other applications may have stringent real-time requirements and prefer to explicitly set a time-out. For example: in realtime detection of hijacking, we set a time-out of few minutes to execute traceroutes as soon as a suspicious BGP event is detected; in the IODA application instead, we relax latency constraints in favor of data completeness and we use a time-out of 30 minutes, since it results in RT routing tables from all the VPs to be available for consumption for 99% of the time bins (we verified it on data from 2014 and 2015).

We designed a system based on meta-data stored in Kafka and multiple *sync servers*, each implementing a different synchronization mechanism: each BGP Corsaro RT plugin writes in the Kafka queue, along with

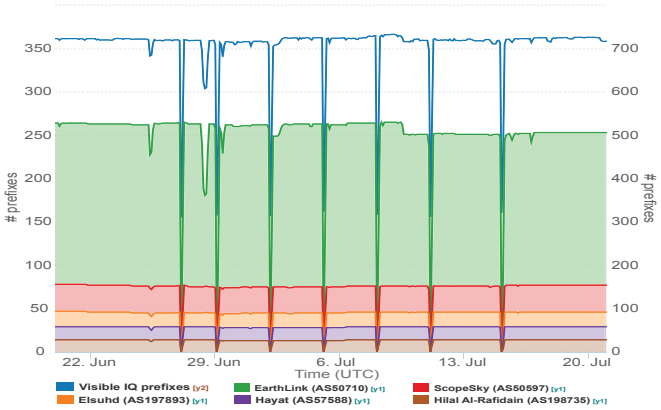


Figure 10: **Visible Iraqi prefixes (June, 20- July, 20 2015).** Number of prefixes observable in BGP that geolocate to Iraq (blue line, $y2$ axis) and number of unique prefixes announced by the 5 top Iraqi providers (stacked lines, $y1$ axis). An observable series of ≈3-hour outages starts on June 27, and ends on July 15. According to the media, the local government ordered complete shutdowns of the Internet service in the country.

the routing tables, indexing meta-data; such meta-data is monitored by the sync servers, which based on the synchronization criterion they implement, inject meta-data into their own topic in the Kafka queue to mark data as ready for consumption. By using Kafka, the resulting system is horizontally scalable (since Kafka supports distributing data across many nodes) and robust (e.g., due to data replication). In addition, since sync servers only handle lightweight meta-data which have a small memory footprint, they do not affect scalability.

6.2.4 *Consumers*

Consumers implement routines that analyze the routing tables retrieved from Kafka to perform event detection, extraction of statistics to output as time series etc. We developed two consumers for near-realtime detection of per-country and per-AS outages. Both consumers select the prefixes observed by full-feed VPs and monitor the visibility of these prefixes by computing the number of prefixes geo-located to each country and announced by each AS. The consumers store this data into a time series monitoring system supporting automated change-point detection and data visualization.

Figure 10 shows data from the per-country and per-AS outages consumers over a period of 1 month, (June 20 to July 20, 2015), selecting prefix visibility associated with Iraq and five of the biggest Iraqi ISPs. The noticeable drops reflect a sequence of country-wide Internet outages that the government ordered in conjunction with the ministerial preparatory exams [8, 29, 32].

7. CONCLUSIONS

BGPStream targets a broad range of applications and users. We hope that it will enable novel analyses, development of new tools, educational opportunities, as

well as feedback and contributions to our platform. In addition, since code and scripts using BGPStream embed the definition of the public data sources used for an experiment, BGPStream significantly eases the reproducibility of experimental results.

BGPStream development is part of a collaborative effort with other researchers and data providers, such as Cisco, RouteViews and BGPMon, to coordinate progress in this space [17]. We plan to release new features in the near future, including support for more data formats (e.g., JSON exports from ExaBGP [31], OpenBMP [30]). In particular, adding native support for OpenBMP will enable processing of streams sourced directly from BGP routers.

Acknowledgements

We would like to thank the reviewers for their insightful comments and in particular the shepherd, Dave Choffnes, for many suggestions on further improving the paper. This work was supported by National Science Foundation grants CNS-1228994 and CNS-1423659. This work was also supported by the Department of Homeland Security Science and Technology Directorate, Cyber Security Division (DHS S&T/CSD), via contract N66001-12-C-0130 and grant FA8750-12-2-0314, cooperative agreement FA8750-12-2-0326. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by NSF grant number ACI-1053575. Danilo Giordano's contribution to this research has been partially funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-129, BigDAMA.

8. REFERENCES

[1] Apache Kafka. http://kafka.apache.org/, 2015.
[2] Apache Spark. http://spark.apache.org/, 2015.
[3] Colorado State University. BGPmon. http://www.bgpmon.io/, 2015.
[4] E. Aben. Has the Routability of Longer-than-/24 Prefixes Changed? https://labs.ripe.net/Members/emileaben/has-the-routability-of-longer-than-24-prefixes-changed, September 2015.
[5] S. Anisseh. Internet Topology Characterizationon on AS Level. Master's thesis, KTH, School of Electrical Engineering, 10 2012.
[6] ARBOR Networks. ATLAS Q2 2015 Global DDoS Attack Trends. https://resources.arbornetworks.com/h/i/110843942-atlas-q2-2015-global-ddos-attack-trends, 2014.
[7] G. D. Battista, M. Rimondini, and G. Sadolfo. Monitoring the status of MPLS VPN and VPLS based on BGP signaling information. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 237–244. IEEE, 2012.
[8] D. Bernard. Iraqi Internet Experiencing 'Strange' Outages. http://www.voanews.com/content/iraqi-internet-experiencing-strange-outages/2921135.html, 2015.
[9] L. Blunk, M. Karir, and C. Labovitz. Multi-Threaded Routing Toolkit (MRT) Routing Information Export Format. RFC 6396 (Proposed Standard), Oct. 2011.
[10] R. Bush, O. Maennel, M. Roughan, and S. Uhlig. Internet optometry: assessing the broken glasses in internet reachability. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 242–253. ACM, 2009.
[11] CAIDA. BGPStream. https://github.com/CAIDA/bgpstream, 2016.
[12] CAIDA. BGPStream. https://bgpstream.caida.org/, 2016.
[13] CAIDA. CAIDA BGP Hackathon 2016. https://www.caida.org/workshops/bgp-hackathon/1602/, 2016.
[14] CAIDA. Supplemental data: BGPStream: a software framework for live and historical BGP data analysis. http://www.caida.org/publications/papers/2016/bgpstream/supplemental/, 2016.
[15] R. Chandra, P. Traina, and T. Li. BGP Communities Attribute. RFC 1997 (Proposed Standard), Aug. 1996. Updated by RFC 7606.
[16] Cisco Systems. Remotely Triggered Black Holed Filtering. http://www.cisco.com/c/dam/en_us/about/security/intelligence/blackhole.pdf, 2005.
[17] k. claffy. The 8th Workshop on Active Internet Measurements (AIMS8) Report. *ACM SIGCOMM Computer Communication Review (CCR)*, Jul 2016.
[18] M. Cosovic, S. Obradovic, and L. Trajkovic. Performance evaluation of BGP anomaly classifiers. In *Digital Information, Networking, and Wireless Communications (DINWC), 2015 Third International Conference on*, pages 115–120. IEEE, 2015.
[19] J. Cowie. The New Threat: Targeted Internet Traffic Misdirection. http://research.dyn.com/2013/11/mitm-internet-hijacking/, 2013.
[20] A. Dainotti. HIJACKS: Detecting and Characterizing Internet Traffic Interception based on BGP Hijacking. http://www.caida.org/funding/hijacks/, 2014. Funding source: NSF CNS-1423659.
[21] A. Dainotti. North Korean Internet outages observed. http://blog.caida.org/best_available_data/2014/12/23/north-korean-internet-outages-observed/, 2014.
[22] A. Dainotti and V. Asturiano. Under the Telescope: Time Warner Cable Internet Outage. http://blog.caida.org/best_available_data/2014/08/29/under-the-telescope-time-warner-cable-internet-outage/, 2014.
[23] A. Dainotti and K. Claffy. Detection and analysis of large-scale Internet infrastructure outages (IODA). http://www.caida.org/funding/ioda/, 2012. Funding source: NSF CNS-1228994.
[24] A. Dainotti, A. King, C. Orsini, and V. Asturiano. BGPStream: a framework for BGP data analysis. https://ripe70.ripe.net/presentations/55-bgpstream.pdf, 2015.
[25] C. Dietzel, A. Feldmann, and T. King. Blackholing at ixps: On the effectiveness of ddos mitigation in the wild. In *Passive and Active Network Measurement (PAM)*, pages 319–332. Springer, 2016.
[26] B. Donnet and O. Bonaventure. On BGP communities. *SIGCOMM Comput. Commun. Rev.*, 38(2):55–59, 2008.

[27] J. Durand, I. Pepelnjak, and G. Doering. BGP Operations and Security. RFC 7454 (Best Current Practice), Feb. 2015.

[28] Dyn Research. Routing alarms. http://research.dyn.com/products/routing-alarms/.

[29] Dyn Research. Iraq has had 12 govt-directed Internet blackouts since 27-Jun. https://twitter.com/DynResearch/status/629393185517666305, 2015.

[30] T. Evens. OpenBMP. http://http://www.openbmp.org/, 2015.

[31] Exa-Networks. ExaBGP. https://github.com/Exa-Networks/exabgp, 2015.

[32] S. Gallagher. Iraqi government shut down Internet to prevent exam cheating? http://arstechnica.com/tech-policy/2015/06/iraqi-government-shut-down-internet-to-prevent-exam-cheating/, 2015.

[33] L. Gao and F. Wang. The extent of as path inflation by routing policies. In *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, volume 3, pages 2180–2184. IEEE, 2002.

[34] V. Giotsas, M. Luckie, B. Huffaker, et al. Inferring complex as relationships. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 23–30. ACM, 2014.

[35] X. Hu and Z. M. Mao. Accurate real-time identification of ip prefix hijacking. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 3–17. IEEE, 2007.

[36] Q. Jacquemart, G. Urvoy-Keller, and E. Biersack. A longitudinal study of bgp moas prefixes. In *Traffic Monitoring and Analysis*, pages 127–138. Springer, 2014.

[37] E. Karaarslan, A. G. Perez, and C. Siaterlis. Recreating a Large-Scale BGP Incident in a Realistic Environment. In *Information Sciences and Systems 2013*, pages 349–357. Springer, 2013.

[38] D. E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[39] W. Kumari and D. McPherson. Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF). RFC 5635 (Informational), Aug. 2009.

[40] C. Labovitz, A. Ahuja, S. Venkatachary, and R. Wattenhofer. The Impact of Internet Policy and Topology on Delayed Routing Convergence. In *20th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, April 2001.

[41] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang. Phas: A prefix hijack alert system. In *Proceedings of the 15th Conference on USENIX Security Symposium*, 2006.

[42] M. Luckie. Spurious routes in public bgp data. *ACM SIGCOMM Computer Communication Review*, 44(3):14–21, 2014.

[43] M. Luckie, B. Huffaker, A. Dhamdhere, V. Giotsas, and k claffy. AS relationships, customer cones, and validation. In *IMC*, Oct. 2013.

[44] A. Lutu, M. Bagnulo, J. Cid-Sueiro, and O. Maennel. Separating wheat from chaff: Winnowing unintended prefixes using machine learning. In *INFOCOM, 2014 Proceedings IEEE*, pages 943–951. IEEE, 2014.

[45] D. Madory. The Vast World of Fraudulent Routing. http://research.dyn.com/2015/01/vast-world-of-fraudulent-routing/, 2015.

[46] R. Mazloum, M.-O. Buob, J. Auge, B. Baynat, D. Rossi, and T. Friedman. Violation of interdomain routing assumptions. In *Passive and Active Measurement*, pages 173–182. Springer, 2014.

[47] NetworkX Developers. NetworkX. https://networkx.github.io, 2015.

[48] U. of Oregon. Route Views Project. http://www.routeviews.org/, 2015.

[49] PCH. Packet Clearing House. http://www.pch.net/, 2015.

[50] Quagga. Quagga Routing Software Suite. http://www.nongnu.org/quagga/, 2015.

[51] B. Quoitin, C. Pelsser, L. Swinnen, O. Bonaventure, and S. Uhlig. Interdomain traffic engineering with bgp. *Communications Magazine, IEEE*, 41(5):122–128, 2003.

[52] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), Jan. 2006. Updated by RFCs 6286, 6608, 6793, 7606, 7607.

[53] P. Richter. Classification of origin AS behavior based on BGP update streams. Master's thesis, Technische Universitat Berlin, 2010. Bachelor Thesis.

[54] RIPE NCC. libBGPdump. https://bitbucket.org/ripencc/bgpdump, 2015.

[55] RIPE NCC. RIPE Atlas: A Global Internet Measurement Network. *The Internet Protocol Journal*, 18(3), September 2015.

[56] RIPE NCC. Routing Information Service (RIS). https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris, 2015.

[57] D. Schatzmann, B. Plattner, and W. Mühlbauer. Identification of Connectivity Issues in Large Networks using Data Plane Information.

[58] J. Scudder, R. Fernando, and S. Stuart. BGP Monitoring Protocol. Internet-Draft draft-ietf-grow-bmp-14.txt, IETF Secretariat, Aug. 2015.

[59] R. Steenbergen and T. Scholl. BGP Communities: A Guide for Service Provider Networks . NANOG 40, Bellevue, Washington, June 2007.

[60] C. Q. Sun and P. F. Ding. Optimization Techniques of Traceroute Measurement Based on BGP Routing Table. In *Applied Mechanics and Materials*, volume 303, pages 2062–2067. Trans Tech Publ, 2013.

[61] D. Turk. Configuring BGP to Block Denial-of-Service Attacks. RFC 3882 (Informational), Sept. 2004.

[62] M. Wählisch, O. Maennel, and T. C. Schmidt. Towards detecting bgp route hijacking using the rpki. *ACM SIGCOMM Computer Communication Review*, 42(4):103–104, 2012.

[63] H. Yan, R. Oliveira, K. Burnett, D. Matthews, L. Zhang, and D. Massey. BGPmon: A real-time, scalable, extensible monitoring system. In *CATCH'09. Cybersecurity Applications & Technology*, pages 212–223. IEEE, 2009.

[64] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. An analysis of bgp multiple origin as (moas) conflicts. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, IMW '01, pages 31–35, New York, NY, USA, 2001. ACM.